

Ramakrishna Mission Vidyamandira

(An Autonomous College Under University of Calcutta)

MSc. Computer Science & Machine Intelligence,

Semester III, 2022

Paper: CSMI P 5 P

**Deep Learning Lab assignment submission**

|  |
| --- |
| **Submitted by** |
| Class Roll Number: 531  Registration Number: A04-1122-0566-21  Post-Graduation  3rd Semester  Batch: 2021-23 |

**INDEX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SL. NO.** | **ASSIGNMNET STATEMENT** | **D-O-A** | **D-O-S** | **SIGNATURE** |
| 1 | Implement perceptron with 2-bit binary input (AND, OR, XOR, EX-OR, NAND, NOR). | 25/07/22 | 07/12/22 |  |
| 2 | Implement multilayer perceptron with single hidden layer with one output unit. | 25/07/22 | 07/12/22 |  |
| 3 | Implement multilayer perceptron with single hidden layer with multiple output unit. | 25/07/22 | 07/12/22 |  |
| 4 | Implement multilayer perceptron with multiple hidden layers with one output unit. | 25/07/22 | 07/12/22 |  |
| 5 | Implement multilayer perceptron with multiple hidden layers with multiple output unit. | 25/07/22 | 07/12/22 |  |
| 6 | Implement Fashion MNIST dataset with new test image data. | 24/08/22 | 07/12/22 |  |
| 7 | Implement CNN with CIFAR10 dataset. | 16/11/22 | 07/12/22 |  |

**ASSIGNMENT NO.: 01 Date: 25/07/2022**

**Problem Statement:**

Implement perceptron with 2-bit binary input (AND, OR, XOR, EX-OR, NAND, NOR).

**Source Code:**

# *Python program to i*mplement perceptron with 2-bit binary input (AND, OR, XOR, EX-NOR, NAND, NOR).

import numpy as np

import matplotlib.pyplot as plt

# Fuction for AND Operation

def AND\_Operation(input\_val):

    wAND = np.array([1, 1])

    bAND = -1.5

    return Perceptron(input\_val, wAND, bAND)

# Fuction for OR Operation

def OR\_Operation(input\_val):

    wOR = np.array([1, 1])

    bOR = -0.5

    return Perceptron(input\_val, wOR, bOR)

# Fuction for NOT Operation

def NOT\_Operation(input\_val):

    wNOT = -1

    bNOT = 0.5

    return Perceptron(input\_val, wNOT, bNOT)

# Fuction for XOR Operation

def XOR\_Operation(input\_val):

    y1 = AND\_Operation(input\_val)

    y2 = OR\_Operation(input\_val)

    y3 = NOT\_Operation(y1)

    final\_x = np.array([y2, y3])

    return AND\_Operation(final\_x)

# Fuction for XNOR Operation

def XNOR\_Operation(input\_val):

    y1 = OR\_Operation(input\_val)

    y2 = AND\_Operation(input\_val)

    y3 = NOT\_Operation(y1)

    final\_x = np.array([y2, y3])

    return OR\_Operation(final\_x)

# Fuction for NAND Operation

def NAND\_Operation(input\_val):

    output\_AND = AND\_Operation(input\_val)

    return NOT\_Operation(output\_AND)

# Fuction for NOR Operation

def NOR\_Operation(input\_val):

    output\_OR = OR\_Operation(input\_val)

    return NOT\_Operation(output\_OR)

# Fuction for Perceptron

def Perceptron(input\_val, weight, bias):

    sum = np.dot(input\_val, weight) + bias

    pre\_result = signum(sum)

    return pre\_result

# Fuction for Activation function

def signum(sum):

    return 1 if (sum > 0) else 0

# Fuction for choice 1

def AND():

    for \_ in range(4):

        x1 = int(input(" Enter the 1st input(0/1): "))

        x2 = int(input(" Enter the 2nd input(0/1): "))

        val = np.array([x1,x2])

        print(f"  AND({x1}, {x2}) = {AND\_Operation(val)}")

        plt.title('Classification Data of 2 input AND logic (Red-0, Black-1)')

        if AND\_Operation(val) == 1:

            plt.plot(val[0], val[1], 'o', color = 'black')

        else:

            plt.plot(val[0], val[1], 'o', color = 'red')

    plt.show()

# Fuction for choice 2

def OR():

    for \_ in range(4):

        x1 = int(input(" Enter the 1st input(0/1): "))

        x2 = int(input(" Enter the 2nd input(0/1): "))

        val = np.array([x1,x2])

        print(f"  OR({x1}, {x2}) = {OR\_Operation(val)}")

        plt.title('Classification Data of 2 input OR logic (Red-0, Black-1)')

        if OR\_Operation(val) == 1:

            plt.plot(val[0], val[1], 'o', color = 'black')

        else:

            plt.plot(val[0], val[1], 'o', color = 'red')

    plt.show()

# Fuction for choice 3

def XOR():

    for \_ in range(4):

        x1 = int(input(" Enter the 1st input(0/1): "))

        x2 = int(input(" Enter the 2nd input(0/1): "))

        val = np.array([x1,x2])

        print(f"  XOR({x1}, {x2}) = {XOR\_Operation(val)}")

        plt.title('Classification Data of 2 input XOR logic (Red-0, Black-1)')

        if XOR\_Operation(val) == 1:

            plt.plot(val[0], val[1], 'o', color = 'black')

        else:

            plt.plot(val[0], val[1], 'o', color = 'red')

    plt.show()

# Fuction for choice 4

def XNOR():

    for \_ in range(4):

        x1 = int(input(" Enter the 1st input(0/1): "))

        x2 = int(input(" Enter the 2nd input(0/1): "))

        val = np.array([x1,x2])

        print(f"  XNOR({x1}, {x2}) = {XNOR\_Operation(val)}")

        plt.title('Classification Data of 2 input EX-NOR logic (Red-0, Black-1)')

        if XNOR\_Operation(val) == 1:

            plt.plot(val[0], val[1], 'o', color = 'black')

        else:

            plt.plot(val[0], val[1], 'o', color = 'red')

    plt.show()

# Fuction for choice 5

def NAND():

    for \_ in range(4):

        x1 = int(input(" Enter the 1st input(0/1): "))

        x2 = int(input(" Enter the 2nd input(0/1): "))

        val = np.array([x1,x2])

        print(f"  NAND({x1}, {x2}) = {NAND\_Operation(val)}")

        plt.title('Classification Data of 2 input NAND logic (Red-0, Black-1)')

        if NAND\_Operation(val) == 1:

            plt.plot(val[0], val[1], 'o', color = 'black')

        else:

            plt.plot(val[0], val[1], 'o', color = 'red')

    plt.show()

# Fuction for choice 6

def NOR():

    for \_ in range(4):

        x1 = int(input(" Enter the 1st input(0/1): "))

        x2 = int(input(" Enter the 2nd input(0/1): "))

        val = np.array([x1,x2])

        print(f"  NOR({x1}, {x2}) = {NOR\_Operation(val)}")

        plt.title('Classification Data of 2 input NOR logic (Red-0, Black-1)')

        if NOR\_Operation(val) == 1:

            plt.plot(val[0], val[1], 'o', color = 'black')

        else:

            plt.plot(val[0], val[1], 'o', color = 'red')

    plt.show()

# Fuction for wrong choice

def nothing():

    print("  You choose a wrong choice.")

# Fuction for switch case

def switch\_case(choice):

    return switcher.get(choice,nothing)()

# Main fuction to test above

if \_\_name\_\_ == "\_\_main\_\_":

    print("\tChoice: 1 for AND Operation")

    print("\tChoice: 2 for OR Operation")

    print("\tChoice: 3 for XOR Operation")

    print("\tChoice: 4 for EX-NOR Operation")

    print("\tChoice: 5 for NAND Operation")

    print("\tChoice: 6 for NOR Operation")

    print("\tChoice: 0 for exit from execution\n")

    switcher = {

        1: AND,

        2: OR,

        3: XOR,

        4: XNOR,

        5: NAND,

        6: NOR

        }

    ch = int(input("Enter your Choice: "))

    while(ch):

        switch\_case(ch)

        ch = int(input("\nEnter your Choice: "))

**Output:**

Choice: 1 for AND Operation

Choice: 2 for OR Operation

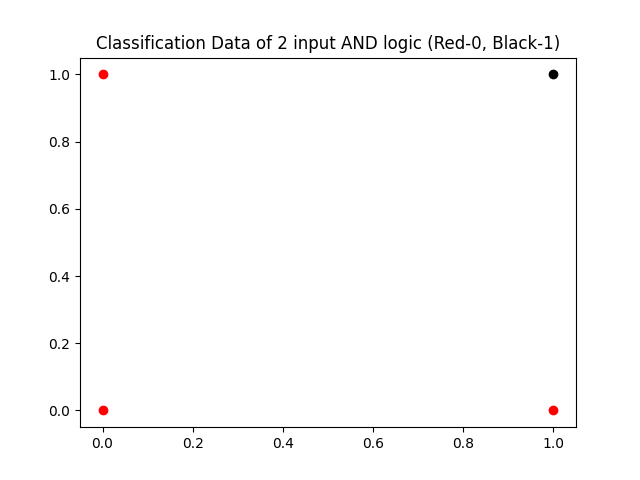
Choice: 3 for XOR Operation

Choice: 4 for EX-NOR Operation

Choice: 5 for NAND Operation

Choice: 6 for NOR Operation

Choice: 0 for exit from execution

Enter your Choice: 1

Enter the 1st input(0/1): 0

Enter the 2nd input(0/1): 0

AND(0, 0) = 0

Enter the 1st input(0/1): 0

Enter the 2nd input(0/1): 1

AND(0, 1) = 0

Enter the 1st input(0/1): 1

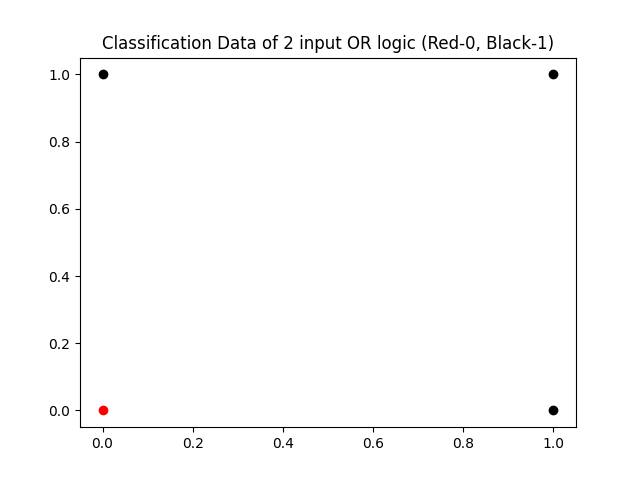
Enter the 2nd input(0/1): 0

AND(1, 0) = 0

Enter the 1st input(0/1): 1

Enter the 2nd input(0/1): 1

AND(1, 1) = 1



Enter your Choice: 2

Enter the 1st input(0/1): 0

Enter the 2nd input(0/1): 0

OR(0, 0) = 0

Enter the 1st input(0/1): 0

Enter the 2nd input(0/1): 1

OR(0, 1) = 1

Enter the 1st input(0/1): 1

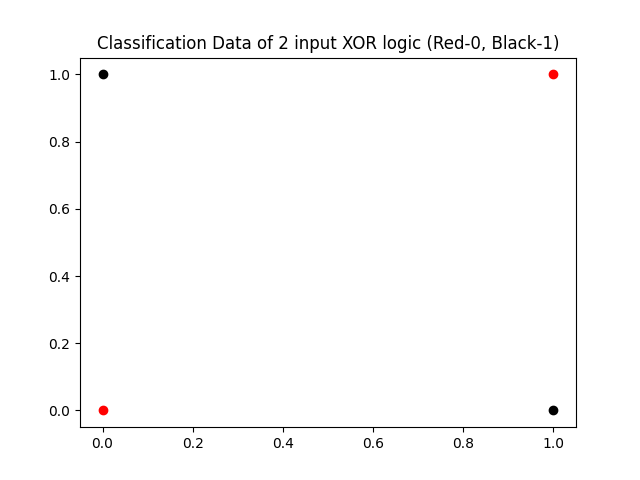
Enter the 2nd input(0/1): 0

OR(1, 0) = 1

Enter the 1st input(0/1): 1

Enter the 2nd input(0/1): 1

OR(1, 1) = 1

Enter your Choice: 3

Enter the 1st input(0/1): 0

Enter the 2nd input(0/1): 0

XOR(0, 0) = 0

Enter the 1st input(0/1): 0

Enter the 2nd input(0/1): 1

XOR(0, 1) = 1

Enter the 1st input(0/1): 1

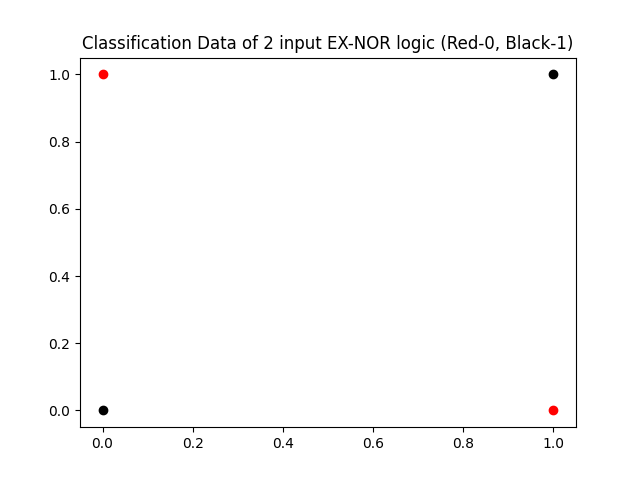
Enter the 2nd input(0/1): 0

XOR(1, 0) = 1

Enter the 1st input(0/1): 1

Enter the 2nd input(0/1): 1

XOR(1, 1) = 0



Enter your Choice: 4

Enter the 1st input(0/1): 0

Enter the 2nd input(0/1): 0

XNOR(0, 0) = 1

Enter the 1st input(0/1): 0

Enter the 2nd input(0/1): 1

XNOR(0, 1) = 0

Enter the 1st input(0/1): 1

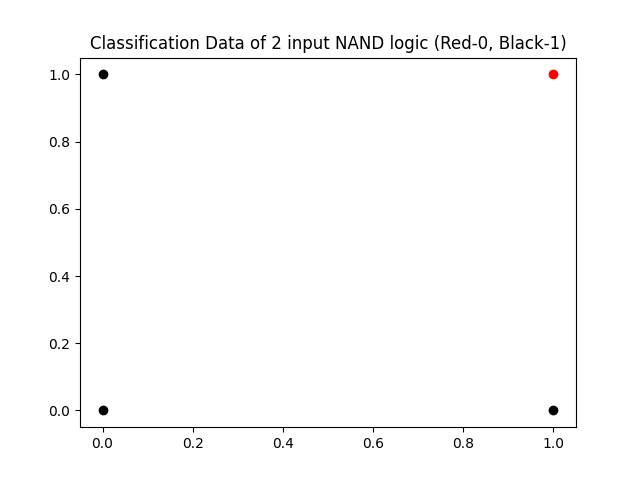
Enter the 2nd input(0/1): 0

XNOR(1, 0) = 0

Enter the 1st input(0/1): 1

Enter the 2nd input(0/1): 1

XNOR(1, 1) = 1

Enter your Choice: 5

Enter the 1st input(0/1): 0

Enter the 2nd input(0/1): 0

NAND(0, 0) = 1

Enter the 1st input(0/1): 0

Enter the 2nd input(0/1): 1

NAND(0, 1) = 1

Enter the 1st input(0/1): 1

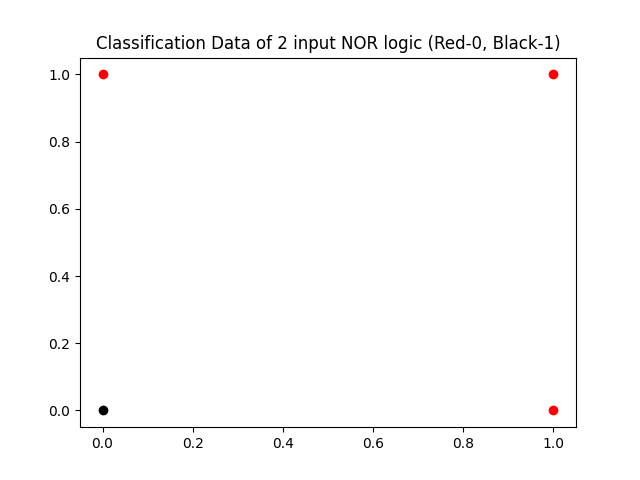
Enter the 2nd input(0/1): 0

NAND(1, 0) = 1

Enter the 1st input(0/1): 1

Enter the 2nd input(0/1): 1

NAND(1, 1) = 0

Enter your Choice: 6

Enter the 1st input(0/1): 0

Enter the 2nd input(0/1): 0

NOR(0, 0) = 1

Enter the 1st input(0/1): 0

Enter the 2nd input(0/1): 1

NOR(0, 1) = 0

Enter the 1st input(0/1): 1

Enter the 2nd input(0/1): 0

NOR(1, 0) = 0

Enter the 1st input(0/1): 1

Enter the 2nd input(0/1): 1

NOR(1, 1) = 0

Enter your Choice: 7

You choose a wrong choice.

Enter your Choice: 0

**ASSIGNMENT NO.: 02 Date:** **25/07/2022**

**Problem Statement:**

Implement multilayer perceptron with single hidden layer with one output unit.

**Source Code:**

# *Python program to i*mplement multilayer perceptron with single hidden layer with one # output unit.

import numpy as np

def Operation(input\_val, weight1, weight2, bias1, bias2):

hidout = hiddenLayer(input\_val, weight1, bias1)

pred = outputLayer(hidout, weight2, bias2)

return hidout, pred

def hiddenLayer(val, weight, bias):

sum = np.dot(val, weight) + bias

z1 = sigmoid(sum)

return z1

def outputLayer(val, weight, bias):

sum = np.dot(val, weight) + bias

pre\_result = sigmoid(sum)

return pre\_result

def sigmoid(x):

return 1/(1 + np.exp(-x))

def backPropagation(op\_size, trgt, hd\_size, learning\_rate,\

ip\_size, data, W1, W2, bias1, bias2, iterations ):

for itr in range(iterations):

A1, A2 = Operation(data, W1, W2, bias1, bias2)

A1 = A1.flatten()

A2 = A2.flatten()

# Calculatring the error b/w o/p & hidden layer

E2 = []

for k in range(op\_size):

dW2 = A2[k] \* (1 - A2[k])

E2.append((trgt[k] - A2[k]) \* dW2)

# Calculatring the error b/w hidden & i/p layer

E1 = []

for j in range(hd\_size):

delj = np.dot(E2, W2.T)

dW1 = A1[j] \* (1 - A1[j])

E1.append(delj[j] \* dW1)

# Updating the weights & bias b/w hidden and o/p layer

for k in range(op\_size):

for j in range(hd\_size):

W2[j][k] += learning\_rate \* E2[k] \* A1[j]

bias2[k] += learning\_rate \* E2[k]

# Updating the weights & bias b/w i/p and hidden layer

for j in range(hd\_size):

for i in range(ip\_size):

W1[i][j] += learning\_rate \* E1[j] \* data[i]

bias1[j] += learning\_rate \* E1[j]

return A2

# Main fuction to test above

if \_\_name\_\_ == "\_\_main\_\_":

ip\_size = 2

hd\_size = 2

op\_size = 1

itr = 5000

learning\_rate = 0.5

data = np.array([0, 1])

W1 = np.array([[0.6, -0.3],[-0.1, 0.4]])

bias1 = [0.3, 0.5]

W2 = np.array([[0.4], [0.1]])

bias2 = [-0.2]

trgt = [1]

final = backPropagation(op\_size, trgt, hd\_size, learning\_rate,\

ip\_size, data, W1, W2, bias1, bias2, itr )

print(f"Final Output after {itr} iterations =", final)

print("Error =", trgt - final)

**Output:**

Final Output after 5000 iterations = [0.99088949]

Error = [0.00911051]

**ASSIGNMENT NO.: 03 Date: 25/07/2022**

**Problem Statement:**

Implement multilayer perceptron with single hidden layer with multiple output unit.

**Source Code:**

*# Python program to implement multilayer perceptron with single hidden layer with multiple output unit*

import numpy as np

def Operation(input\_val, weight1, weight2, bias1, bias2):

    hidout = hiddenLayer(input\_val, weight1, bias1)

    pred = outputLayer(hidout, weight2, bias2)

    return hidout, pred

def hiddenLayer(val, weight, bias):

    sum = np.dot(val, weight) + bias

    z1 = sigmoid(sum)

    return z1

def outputLayer(val, weight, bias):

    sum = np.dot(val, weight) + bias

    pre\_result = sigmoid(sum)

    return pre\_result

def sigmoid(x):

    return 1/(1 + np.exp(-x))

def backPropagation(op\_size, trgt, hd\_size, learning\_rate,\

    ip\_size, data, W1, W2, bias1, bias2, iterations ):

    for itr in range(iterations):

        A1, A2 = Operation(data, W1, W2, bias1, bias2)

        A1 = A1.flatten()

        A2 = A2.flatten()

*# Calculatring the error b/w o/p & hidden layer*

        E2 = []

        for k in range(op\_size):

            dW2 = A2[k] \* (1 - A2[k])

            E2.append((trgt[k] - A2[k]) \* dW2)

*# Calculatring the error b/w hidden & i/p layer*

        E1 = []

        for j in range(hd\_size):

            delj = np.dot(E2, W2.T)

            dW1 = A1[j] \* (1 - A1[j])

            E1.append(delj[j] \* dW1)

*# Updating the weights & bias b/w  hidden and o/p layer*

        for k in range(op\_size):

            for j in range(hd\_size):

                W2[j][k] += learning\_rate \* E2[k] \* A1[j]

            bias2[k] += learning\_rate \* E2[k]

*# Updating the weights & bias b/w i/p and hidden layer*

        for j in range(hd\_size):

            for i in range(ip\_size):

                W1[i][j] += learning\_rate \* E1[j] \* data[i]

            bias1[j] += learning\_rate \* E1[j]

    return A2

*# Main fuction to test above*

if \_\_name\_\_ == "\_\_main\_\_":

    ip\_size = 2

    hd\_size = 2

    op\_size = 2

    itr = 5000

    learning\_rate = 0.5

    data = np.array([0, 1])

    W1 = np.array([[0.6, -0.3],[-0.1, 0.4]])

    bias1 = [0.3, 0.5]

    W2 = np.array([[0.4, 0.3], [0.1, 0.2]])

    bias2 = [-0.2, -0.4]

    trgt = [1, 0]

    final = backPropagation(op\_size, trgt, hd\_size, learning\_rate,\

        ip\_size, data, W1, W2, bias1, bias2, itr )

    print(f"Final Output after {itr} iteration = ", final)

    print("Error =", trgt - final)

**Output:**

Final Output after 5000 iteration = [0.99124187 0.0090018 ]

Error = [ 0.00875813 -0.0090018 ]

**ASSIGNMENT NO.: 04 Date: 25/07/2022**

**Problem Statement:**

Implement multilayer perceptron with multiple hidden layers with one output unit.

**Source Code:**

*# Python program to implement multilayer perceptron with multiple hidden layers with one output unit.*

import keras

import numpy as np

import pandas as pd

df = pd.read\_csv("3\_Input\_XOR.csv",)

df.head()

print("Shape of df:",df.shape)

xs = df.iloc[: , :-1]

ys = df.iloc[: , -1]

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(xs, ys, test\_size=0.3, random\_state=42)

print("Shape of X\_train:", X\_train.shape)

# 2 hidden layers 2 output unit

model = keras.Sequential([

keras.layers.Dense(3, input\_shape=(3,), activation='relu'),

keras.layers.Dense(2, activation='relu'),

keras.layers.Dense(1, activation='sigmoid')

])

opt = keras.optimizers.Adam(learning\_rate=0.09)

model.compile(

optimizer= opt,

loss='binary\_crossentropy',

metrics=['accuracy'])

model.fit(X\_train, y\_train, epochs=50)

model.evaluate(X\_test, y\_test)

model.predict(X\_test)[:5]

y\_test.head()

**Output:**

x1 x2 x3 xor(x)

0 0 0 0 0

1 0 0 1 1

2 0 1 0 1

3 0 1 1 0

4 1 0 0 1

Shape of df: (34, 4)

Shape of X\_train: (23, 3)

Epoch 1/50

1/1 [==============================] - 0s 381ms/step - loss: 0.7010 - accuracy: 0.4783

Epoch 2/50

1/1 [==============================] - 0s 6ms/step - loss: 0.6542 - accuracy: 0.5652

Epoch 3/50

1/1 [==============================] - 0s 6ms/step - loss: 0.6473 - accuracy: 0.5217

…

Epoch 47/50

1/1 [==============================] - 0s 5ms/step - loss: 0.0306 - accuracy: 1.0000

Epoch 48/50

1/1 [==============================] - 0s 6ms/step - loss: 0.0265 - accuracy: 1.0000

Epoch 49/50

1/1 [==============================] - 0s 6ms/step - loss: 0.0230 - accuracy: 1.0000

Epoch 50/50

1/1 [==============================] - 0s 6ms/step - loss: 0.0221 - accuracy: 1.0000

<keras.callbacks.History at 0x24a9d634430>

1/1 [==============================] - 0s 114ms/step - loss: 0.0449 - accuracy: 1.0000

[0.044938549399375916, 1.0]

1/1 [==============================] - 0s 70ms/step

array([[0.07276429],

[0.9583846 ],

[0.9941003 ],

[0.07276429],

[0.03316279]], dtype=float32)

15 0

19 1

27 1

26 0

8 0

Name: xor(x), dtype: int64

**ASSIGNMENT NO.: 05 Date: 25/07/2022**

**Problem Statement:**

Implement multilayer perceptron with multiple hidden layers with multiple output unit.

**Source Code:**

*# Python program to implement multilayer perceptron with multiple hidden layers with multiple output unit.*

import keras

import numpy as np

import pandas as pd

df = pd.read\_csv("3\_Input\_XOR.csv",)

print("\tdf:\n",df)

# Do one hot encoding to get multiple output

df1= pd.get\_dummies(data=df, columns=['xor(x)'])

print(("\tdf1:\n",df1)

xs = df1.iloc[: , :-2]

ys = df1.iloc[: , -2:]

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(xs, ys, test\_size=0.4, random\_state=5)

print("Shape of X\_train:", X\_train.shape)

# 2 hidden layers 2 output unit

model = keras.Sequential([

keras.layers.Dense(3, input\_shape=(3,), activation='relu'),

keras.layers.Dense(2, activation='relu'),

keras.layers.Dense(2, activation='softmax')

])

opt = keras.optimizers.Adam(learning\_rate=0.03)

model.compile(

optimizer= opt,

loss='mse',

metrics=['accuracy'])

model.fit(X\_train, y\_train, epochs=45)

model.evaluate(X\_test, y\_test)

model.predict(X\_test)[:5]

y\_test.head()

**Output:**

df:

x1 x2 x3 xor(x)

0 0 0 0 0

1 0 0 1 1

2 0 1 0 1

…

31 0 1 1 0

32 1 0 0 1

33 1 0 1 0

df1:

x1 x2 x3 xor(x)\_0 xor(x)\_1

0 0 0 0 1 0

1 0 0 1 0 1

2 0 1 0 0 1

…

31 0 1 1 1 0

32 1 0 0 0 1

33 1 0 1 1 0

Shape of X\_train: (20, 3)

Epoch 1/45

1/1 [==============================] - 0s 330ms/step - loss: 0.2743 - accuracy: 0.4500

Epoch 2/45

1/1 [==============================] - 0s 5ms/step - loss: 0.2574 - accuracy: 0.5500

Epoch 3/45

1/1 [==============================] - 0s 5ms/step - loss: 0.2455 - accuracy: 0.5500

…

Epoch 43/45

1/1 [==============================] - 0s 6ms/step - loss: 0.1335 - accuracy: 0.9000

Epoch 44/45

1/1 [==============================] - 0s 5ms/step - loss: 0.1304 - accuracy: 0.9000

Epoch 45/45

1/1 [==============================] - 0s 6ms/step - loss: 0.1265 - accuracy: 0.9000

<keras.callbacks.History at 0x226c1c97550>

1/1 [==============================] - 0s 98ms/step - loss: 0.1636 - accuracy: 0.8571

[0.16364876925945282, 0.8571428656578064]

1/1 [==============================] - 0s 48ms/step

array([[0.3821941 , 0.6178059 ],

[0.86676633, 0.13323364],

[0.5613875 , 0.43861246],

[0.5613875 , 0.43861246],

[0.80010027, 0.19989972]], dtype=float32)

|  |  | **xor(x)\_0** | **xor(x)\_1** |
| --- | --- | --- | --- |
| 30 |  | 0 | 1 |
| 21 |  | 1 | 0 |
| 3 |  | 1 | 0 |
| 18 |  | 1 | 0 |
| 20 |  | 1 | 0 |

**ASSIGNMENT NO.: 06 Date: 24/08/2022**

**Problem Statement:**

Implement Fashion Mnist dataset with new test image data.

**Source Code:**

*#* *Python program to implement Fashion Mnist dataset with new test image data.*

import keras

import numpy as np

import matplotlib.pyplot as plt

(train\_images, train\_labels), \_ = keras.datasets.fashion\_mnist.load\_data()

# Scale these values to a range of 0 to 1 before feeding them to the neural network model.

train\_images = train\_images / 255.0

model = keras.Sequential([

keras.layers.Flatten(input\_shape=(28, 28)),

keras.layers.Dense(128, activation='relu'),

keras.layers.Dense(10, activation='softmax')

])

model.compile(optimizer='adam',

loss='SparseCategoricalCrossentropy',

metrics=['accuracy'])

model.fit(train\_images, train\_labels, epochs=10)

import cv2

img1 = cv2.imread('ankleBoot28x28.jpg')

print(img1.shape)

plt.imshow(img1)

plt.colorbar()

img2 = cv2.resize(img1, (28,28), interpolation = cv2.INTER\_AREA)

img2.shape

img = cv2.cvtColor(img2, cv2.COLOR\_BGR2GRAY)

print(img.shape)

plt.imshow(img)

plt.colorbar()

img3 = img / 255.0

print(img3.shape)

pred = model.predict(img3)

print(pred)

np.argmax(pred)

class\_names[np.argmax(pred)]

**Output:**

Epoch 1/10

1875/1875 [==============================] - 5s 2ms/step - loss: 0.4981 - accuracy: 0.8237

Epoch 2/10

1875/1875 [==============================] - 4s 2ms/step - loss: 0.3718 - accuracy: 0.8647

Epoch 3/10

1875/1875 [==============================] - 4s 2ms/step - loss: 0.3352 - accuracy: 0.8778

Epoch 4/10

1875/1875 [==============================] - 4s 2ms/step - loss: 0.3116 - accuracy: 0.8860

Epoch 5/10

1875/1875 [==============================] - 4s 2ms/step - loss: 0.2926 - accuracy: 0.8920

Epoch 6/10

1875/1875 [==============================] - 4s 2ms/step - loss: 0.2800 - accuracy: 0.8969

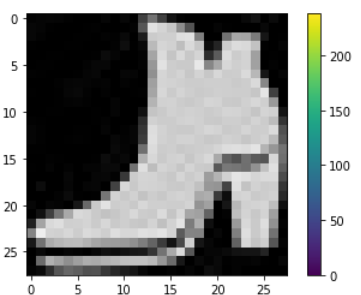
Epoch 7/10

1600/1875 [========================>.....] - ETA: 0s - loss: 0.2659 - accuracy: 0.9014

<keras.callbacks.History at 0x269b57a8430>

(28, 28, 3)

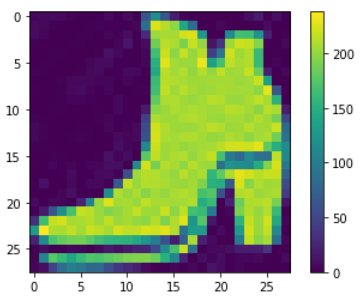
<matplotlib.colorbar.Colorbar at 0x26ab7e5c250>



(28, 28, 3)

(28, 28)

<matplotlib.colorbar.Colorbar at 0x26ab7f47ca0>



(28, 28)

1/1 [==============================] - 0s 19ms/step

array([[6.2257822e-13, 3.3596162e-12, 2.7804211e-16, 6.4195573e-15,

6.1792529e-17, 6.1085865e-07, 2.9868976e-15, 6.1736559e-06,

8.7790859e-14, 9.9999321e-01]], dtype=float32)

9

'Ankle boot'

**ASSIGNMENT NO.: 07 Date: 16/11/2022**

**Problem Statement:**

Implement CNN with CIFAR10 dataset.

**Source Code:**

*# Python program to implement CNN with CIFAR10 dataset.*

# Import TensorFlow and other libraries

import tensorflow as tf

from keras import datasets, layers, models

import matplotlib.pyplot as plt

# Download and prepare the CIFAR10 dataset

(train\_images, train\_labels), (test\_images, test\_labels) = datasets.cifar10.load\_data()

# Normalize pixel values to be between 0 and 1

train\_images, test\_images = train\_images / 255.0, test\_images / 255.0

#Verify the data

class\_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',

'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(10,10))

for i in range(25):

plt.subplot(5,5,i+1)

plt.xticks([])

plt.yticks([])

plt.grid(False)

plt.imshow(train\_images[i])

# The CIFAR labels happen to be arrays,

# which is why you need the extra index

plt.xlabel(class\_names[train\_labels[i][0]])

plt.show()

# Create the convolutional base

model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))

# Let's display the architecture of your model so far:

model.summary()

# Add Dense layers on top

model.add(layers.Flatten())

model.add(layers.Dense(64, activation='relu'))

model.add(layers.Dense(10))

# Here's the complete architecture of your model:

model.summary()

#Compile and train the model

model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True),

metrics=['accuracy'])

history = model.fit(train\_images, train\_labels, epochs=10,

validation\_data=(test\_images, test\_labels))

# Evaluate the model

plt.plot(history.history['accuracy'], label='accuracy')

plt.plot(history.history['val\_accuracy'], label = 'val\_accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.ylim([0.5, 1])

plt.legend(loc='lower right')

test\_loss, test\_acc = model.evaluate(test\_images, test\_labels, verbose=2)

print(test\_acc)

pred = model.predict(test\_images)

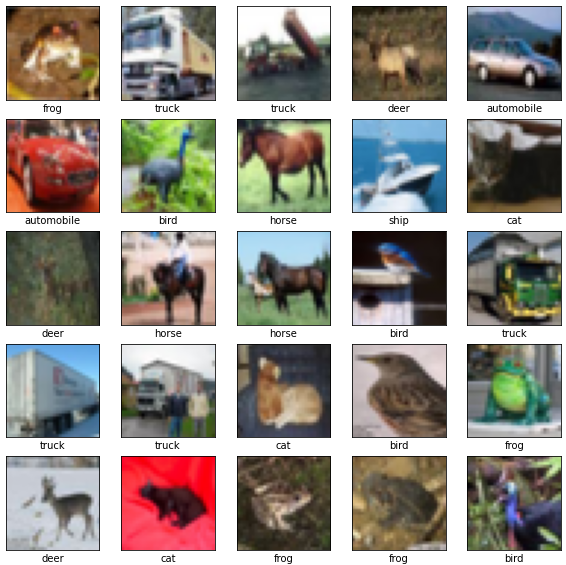
pred[0]

import numpy as np

np.argmax(pred[0])

test\_labels[0]

**Output:**



Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv2d (Conv2D) (None, 30, 30, 32) 896

max\_pooling2d (MaxPooling2D (None, 15, 15, 32) 0

)

conv2d\_1 (Conv2D) (None, 13, 13, 64) 18496

max\_pooling2d\_1 (MaxPooling (None, 6, 6, 64) 0

2D)

conv2d\_2 (Conv2D) (None, 4, 4, 64) 36928

=================================================================

Total params: 56,320

Trainable params: 56,320

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv2d (Conv2D) (None, 30, 30, 32) 896

max\_pooling2d (MaxPooling2D (None, 15, 15, 32) 0

)

conv2d\_1 (Conv2D) (None, 13, 13, 64) 18496

max\_pooling2d\_1 (MaxPooling (None, 6, 6, 64) 0

2D)

conv2d\_2 (Conv2D) (None, 4, 4, 64) 36928

flatten (Flatten) (None, 1024) 0

dense (Dense) (None, 64) 65600

dense\_1 (Dense) (None, 10) 650

=================================================================

Total params: 122,570

Trainable params: 122,570

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Epoch 1/10

1563/1563 [==============================] - 15s 6ms/step - loss: 1.4134 - accuracy: 0.4885 - val\_loss: 1.2173 - val\_accuracy: 0.5637

Epoch 2/10

1563/1563 [==============================] - 10s 6ms/step - loss: 1.1335 - accuracy: 0.5992 - val\_loss: 1.0961 - val\_accuracy: 0.6174

Epoch 3/10

1563/1563 [==============================] - 9s 6ms/step - loss: 0.9793 - accuracy: 0.6584 - val\_loss: 0.9848 - val\_accuracy: 0.6605

Epoch 4/10

1563/1563 [==============================] - 9s 6ms/step - loss: 0.8832 - accuracy: 0.6925 - val\_loss: 0.9485 - val\_accuracy: 0.6697

Epoch 5/10

1563/1563 [==============================] - 9s 6ms/step - loss: 0.8100 - accuracy: 0.7167 - val\_loss: 0.9079 - val\_accuracy: 0.6927

Epoch 6/10

1563/1563 [==============================] - 9s 6ms/step - loss: 0.7514 - accuracy: 0.7374 - val\_loss: 0.8855 - val\_accuracy: 0.6940

Epoch 7/10

1563/1563 [==============================] - 9s 6ms/step - loss: 0.7075 - accuracy: 0.7532 - val\_loss: 0.8531 - val\_accuracy: 0.7105

Epoch 8/10

1563/1563 [==============================] - 9s 6ms/step - loss: 0.6642 - accuracy: 0.7674 - val\_loss: 0.8845 - val\_accuracy: 0.7046

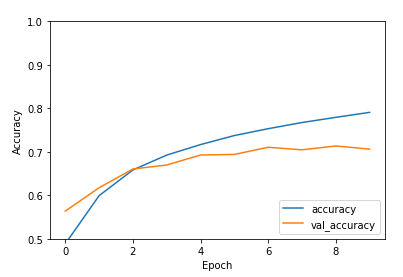
Epoch 9/10

1563/1563 [==============================] - 9s 6ms/step - loss: 0.6282 - accuracy: 0.7793 - val\_loss: 0.8458 - val\_accuracy: 0.7135

Epoch 10/10

1563/1563 [==============================] - 9s 6ms/step - loss: 0.5923 - accuracy: 0.7908 - val\_loss: 0.8780 - val\_accuracy: 0.7062

313/313 - 1s - loss: 0.8780 - accuracy: 0.7062 - 832ms/epoch - 3ms/step



0.7062000036239624

313/313 [==============================] - 1s 2ms/step

array([-2.7867076, -2.1648874, 0.2677088, 3.7510917, -2.1850667,

1.9004017, -1.3009608, -2.0516706, -1.1433753, -1.8290225],

dtype=float32)

3

array([3], dtype=uint8)